Design Theory for Relational DBs: Functional Dependencies, Decompositions, Normal Forms

Introduction to Databases
Manos Papagelis

Thanks to Ryan Johnson, John Mylopoulos, Arnold Rosenbloom and Renee Miller for material in these slides
Database Design Theory

• Guides systematic improvements to database schemas
• General idea:
  – Express constraints on the data
  – Use these to decompose the relations
• Ultimately, get a schema that is in a “normal form”
  – guarantees certain desirable properties
  – “normal” in the sense of conforming to a standard
• The process of converting a schema to a normal form is called normalization
Goal #1: remove redundancy

• Consider this schema

<table>
<thead>
<tr>
<th>Student Name</th>
<th>Student Email</th>
<th>Course</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSC333</td>
<td>Smith</td>
</tr>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSC444</td>
<td>Brown</td>
</tr>
<tr>
<td>Jaspreet</td>
<td>jaspreet@gmail</td>
<td>CSC333</td>
<td>Smith</td>
</tr>
</tbody>
</table>

• What if...
  - Xiao changes email addresses? (*update anomaly*)
  - Xiao drops CSC444? (*deletion anomaly*)
  - Need to create a new course, CSC222 (*insertion anomaly*)

*Multiple relations => exponentially worse*
Goal #2: expressing constraints

• Consider the following sets of schemas:
  Students(utorid, name, email)
  vs.
  Students(utorid, name)
  Emails(utorid, address)

• Consider also:
  House(street, city, value, owner, propertyTax)
  vs.
  House(street, city, value, owner)
  TaxRates(city, value, propertyTax)

Dependencies, constraints are domain-dependent
Overview

- Part I: Functional Dependencies
- Part II: Decompositions
- Part III: Normal Forms
PART 1:
FUNCTIONAL DEPENDENCIES
Functional dependencies

- Let $X, Y$ be sets of attributes from relation $R$
- $X \rightarrow Y$ is an assertion about tuples in $R$
  - Any tuples in $R$ which agree in all attributes of $X$ must also agree in all attributes of $Y$
- “$X$ functionally determines $Y$”
  - Or, “The values of attributes $Y$ are a function of those in $X$”
  - Not necessarily an easy function to compute, mind you
  => Consider $X \rightarrow h$, where $h$ is the hash of attributes in $X$
- Notational conventions
  - “a”, “b”, “c” – specific attributes
  - “A”, “B”, “C” – sets of (unnamed) attributes
  - $abc \rightarrow def$ – same as $\{a,b,c\} \rightarrow \{d,e,f\}$

*Most common to see singletons ($X \rightarrow y$ or $abc \rightarrow d$)*
Rules and principles about FDs

• Rules
  – The splitting/combining rule
  – Trivial FDs
  – The transitive rule

• Algorithms related to FDs
  – the closure of a set of attributes of a relation
  – a minimal basis of a relation
The Splitting/Combining rule of FDs

• Attributes on right independent of each other
  – Consider $a,b,c \rightarrow d,e,f$
  – “Attributes $a$, $b$, and $c$ functionally determine $d$, $e$, and $f$”
    => No mention of $d$ relating to $e$ or $f$ directly

• Splitting rule (Useful to split up right side of FD)
  – $abc \rightarrow def$ becomes $abc \rightarrow d$, $abc \rightarrow e$ and $abc \rightarrow f$

• No safe way to split left side
  – $abc \rightarrow def$ is NOT the same as $ab \rightarrow def$ and $c \rightarrow def$!

• Combining rule (Useful to combine right sides):
  – if $abc \rightarrow d$, $abc \rightarrow e$, $abc \rightarrow f$ holds, then $abc \rightarrow def$ holds
Splitting FDs – example

• Consider the relation and FD
  – EmailAddress(user, domain, firstName, lastName)
  – user, domain -> firstName, lastName

• The following hold
  – user, domain -> firstName
  – user, domain -> lastName

• The following do NOT hold!
  – user -> firstName, lastName
  – domain -> firstName, lastName

Gotcha: “doesn’t hold” = “not all tuples” != “all tuples not”
Trivial FDs

• Not all functional dependencies are useful
  – $A \rightarrow A$ always holds
  – $abc \rightarrow a$ also always holds (right side is subset of left side)

• FD with an attribute on both sides is “trivial”
  – Simplify by removing $L \cap R$ from $R$
    $abc \rightarrow ad$ becomes $abc \rightarrow d$
  – Or, in singleton form, delete trivial FDs
    $abc \rightarrow a$ and $abc \rightarrow d$ becomes just $abc \rightarrow d$
Transitive rule

• The transitive rule holds for FDs
  – Consider the FDs: $a \rightarrow b$ and $b \rightarrow c$; then $a\rightarrow c$ holds
  – Consider the FDs: $ad \rightarrow b$ and $b \rightarrow cd$; then $ad\rightarrow cd$ holds or just $ad\rightarrow c$ (because of the trivial dependency rule)
Identifying functional dependencies

• FDs are **domain knowledge**
  – Intrinsic features of the data you’re dealing with
  – Something you know (or assume) about the data

• Database engine cannot identify FDs for you
  – Designer must specify them as part of schema
  – DBMS can only enforce FDs when told to

• DBMS cannot safely “optimize” FDs either
  – It has only a finite sample of the data
  – An FD constrains the entire domain
Coincidence or FD?

<table>
<thead>
<tr>
<th>ID</th>
<th>Email</th>
<th>City</th>
<th>Country</th>
<th>Surname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1983</td>
<td><a href="mailto:tom@gmail.com">tom@gmail.com</a></td>
<td>Toronto</td>
<td>Canada</td>
<td>Fairgrieve</td>
</tr>
<tr>
<td>8624</td>
<td><a href="mailto:mar@bell.com">mar@bell.com</a></td>
<td>London</td>
<td>Canada</td>
<td>Samways</td>
</tr>
<tr>
<td>9141</td>
<td><a href="mailto:scotty@gmail.com">scotty@gmail.com</a></td>
<td>Winnipeg</td>
<td>Canada</td>
<td>Samways</td>
</tr>
<tr>
<td>1204</td>
<td><a href="mailto:birds@gmail.com">birds@gmail.com</a></td>
<td>Aachen</td>
<td>Germany</td>
<td>Lakemeyer</td>
</tr>
</tbody>
</table>

• What if we try to infer FDs from the data?
  – ID -> email, city, country, surname
  – email -> city, country, surname
  – city -> country
  – surname -> country

*Domain knowledge required to validate FDs*
Keys and FDs

- Consider relation $R$ with attributes $A$

- Superkey
  - Any $S \subseteq A$ s.t. $S \rightarrow A$
  - $\Rightarrow$ Any subset of $A$ which determines all remaining attributes in $A$

- Candidate key (or key)
  - $C \subseteq A$ s.t. $C \rightarrow A$ and $X \rightarrow A$ does not hold for any $X \subset C$
  - $\Rightarrow$ A superkey which contains no other superkeys
  - $\Rightarrow$ Remove any attribute and you no longer have a key

- Primary key
  - The candidate key we use to identify the relation
  - $\Rightarrow$ Always exists, only one allowed, doesn’t matter which $C$ we use

- Prime attribute
  - $\exists$ candidate key $C$ s.t. $x \in C$ (attribute that participates in at least one key)
FD: relaxes the concept of a “key”

- Functional dependency: $X \rightarrow Y$
- Superkey: $X \rightarrow R$
- A superkey must include all remaining attributes of the relation on the RHS (Right-Hand-Side)
- An FD can involve just a subset of them
- Example:
  
  Houses(street, city, value, owner, tax)
  - street, city $\rightarrow$ value, owner, tax (both FD and key)
  - city, value $\rightarrow$ tax (FD only)
Cyclic functional dependencies?

- Attributes on right side of one FD may appear on left side of another!
  - Simple example: assume relation (A, B) & FDs: A -> B, B -> A
  - What does this say about A and B?

- Example
  - studentID -> email    email -> studentID
Geometric view of FDs

- Let $D$ be the domain of tuples in $R$
  - Every possible tuple is a point in $D$

- FD $X$ on $R$ restricts tuples in $R$ to a subset of $D$
  - Points in $D$ which violate $X$ cannot be in $R$

- Example: $D(x,y,z)$
  - $xy \rightarrow z$
    - $z = \text{abs}(x) + \text{abs}(y)$
  - $z \rightarrow x,y$
    - $x=y=\text{abs}(z)/2$

\[ (-1, -1, 2) \quad (1, 1, 2) \quad (2, 2, -4) \]
\[ (0, 0, 1) \quad (1, 1, -2) \quad (2, 2, 4) \]
\[ (1, -1, -2) \quad (1, 2, 3) \quad (3, 2, 1) \]
Inferring functional dependencies

- Problem
  - Given FDs $X_1 \rightarrow a_1$, $X_2 \rightarrow a_2$, etc.
  - Does some FD $Y \rightarrow B$ (not given) also hold?

- Consider the dependencies
  
  $A \rightarrow B$, $B \rightarrow C$
  
  Does $A \rightarrow C$ hold?

  Intuitively, $A \rightarrow C$ also holds
  
  The given FDs entail (imply) it (transitivity rule)

How to prove it in the general case?
Closure test for FDs

- Given attribute set \( A \) and FD set \( F \)
  - Denote \( A_F^+ \) as the closure of \( A \) relative to \( F \)
    \[ A_F^+ = \text{set of all FDs given or implied by } A \]

- Computing the [transitive] closure of \( A \)
  - Start: \( A_F^+ = A \), \( F' = F \)
  - While \( \exists X \in F' \) s.t. LHS(X) \( \subseteq A_F^+ \):
    \[ A_F^+ = A_F^+ \cup \text{RHS}(X) \]
    \[ F' = F' - X \]
  - At end: \( A \rightarrow B \ \forall B \in A_F^+ \)
Closure test – example

- Consider $R(a,b,c,d,e,f)$ with FDs $ab \rightarrow c$, $ac \rightarrow d$, $c \rightarrow e$, $ade \rightarrow f$
- Find $A^+$ if $A = ab$ or find $\{a,b\}^+$

$a b c d e f \rightarrow a b c d e f \rightarrow a b c d e f \rightarrow \{a,b\}^+ = \{a,b,c,d,e,f\}$ or $ab \rightarrow cdef$ -- $ab$ is a candidate key!
### Example: Closure Test

**R(A, B, C, D, E)**

\[ F: AB \rightarrow C \]
\[ A \rightarrow D \]
\[ D \rightarrow E \]
\[ AC \rightarrow B \]

<table>
<thead>
<tr>
<th>X</th>
<th>(X_F^+)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{A, D, E}</td>
</tr>
<tr>
<td>AB</td>
<td>{A, B, C, D, E}</td>
</tr>
<tr>
<td>AC</td>
<td>{A, C, B, D, E}</td>
</tr>
<tr>
<td>B</td>
<td>{B}</td>
</tr>
<tr>
<td>D</td>
<td>{D, E}</td>
</tr>
</tbody>
</table>

Is \(AB \rightarrow E\) entailed by \(F\)? *Yes*

Is \(D \rightarrow C\) entailed by \(F\)? *No*

**Result:** \(X_F^+\) allows us to determine all FDs of the form \(X \rightarrow Y\) entailed by \(F\)
Discarding redundant FDs

- **Minimal basis**: opposite extreme from closure

- Given a set of FDs $F$, want to minimize $F'$ s.t.
  - $F' \subseteq F$
  - $F'$ entails $X \ \forall X \in F$

- **Properties of a minimal basis $F'$**
  - RHS is always singleton
  - If any FD is removed from $F'$, $F'$ is no longer a minimal basis
  - If for any FD in $F'$ we remove one or more attributes from the LHS of $X \in F'$, the result is no longer a minimal basis
Constructing a minimal basis

• Straightforward but time-consuming

1. Split all RHS into singletons
2. \( \forall X \in F', \) test whether \( J = (F'-X)^+ \) is still equivalent to \( F^+ \)

=> Might make \( F' \) too small

3. \( \forall i \in \text{LHS}(X) \ \forall X \in F', \) let \( \text{LHS}(X')=\text{LHS}(X)-i \)
   Test whether \( (F'-X+X')^+ \) is still equivalent to \( F^+ \)

=> Might make \( F' \) too big

4. Repeat (2) and (3) until neither makes progress
Minimal Basis: Example

• Relation R: R(A, B, C, D)
• Defined FDs:
  – F = \{A->AC, B->ABC, D->ABC\}

Find the minimal Basis M of F
Minimal Basis: Example (cont.)

1\textsuperscript{st} Step

- \( H = \{A \rightarrow A, A \rightarrow C, B \rightarrow A, B \rightarrow B, B \rightarrow C, D \rightarrow A, D \rightarrow B, D \rightarrow C\} \)

2\textsuperscript{nd} Step

- A\rightarrow A, B\rightarrow B: \textit{can} be removed as trivial
- A\rightarrow C: \textit{can’t} be removed, as there is no other LHS with A
- B\rightarrow A: \textit{can’t} be removed, because for \( J = H \{B \rightarrow A\} \) is \( B^+ = BC \)
- B\rightarrow C: \textit{can} be removed, because for \( J = H \{B \rightarrow C\} \) is \( B^+ = ABC \)
- D\rightarrow A: \textit{can} be removed, because for \( J = H \{D \rightarrow A\} \) is \( D^+ = DBA \)
- D\rightarrow B: \textit{can’t} be removed, because for \( J = H \{D \rightarrow B\} \) is \( D^+ = DC \)
- D\rightarrow C: \textit{can} be removed, because for \( J = H \{D \rightarrow C\} \) is \( D^+ = DBAC \)

Step outcome => \( H = \{A \rightarrow C, B \rightarrow A, D \rightarrow B\} \)
Minimal Basis: Example (cont.)

3\textsuperscript{rd} Step
- H doesn’t change as all LHS in H are single attributes

4\textsuperscript{th} Step
- H doesn’t change

Minimal Basis: $M = H = \{A \rightarrow C, B \rightarrow A, D \rightarrow B\}$
Minimal Basis: Example 2

• Relation R: R(A, B, C)

• Defined FDs:
  – A->B, A->C, B->C, B->A, C->A, C->B
  – AB->, AC-B, BC->A
  – A->BC
  – A->A

• Possible Minimal Bases:
  – {A->B, B->A, B->C, C->B} or
  – {A->B, B->C, C->A}
  – ...
PART II:
SCHEMA DECOMPOSITION
FDs and redundancy

• Given relation $R$ and FDs $F$
  – $R$ often exhibits anomalies due to redundancy
  – $F$ identifies many (not all) of the underlying problems

• Idea
  – Use $F$ to identify “good” ways to split relations
  – Split $R$ into 2+ smaller relations having less redundancy
  – Split up $F$ into subsets which apply to the new relations (compute the projection of functional dependencies)
Schema decomposition

• Given relation R and FDs F
  – Split R into $R_i$ s.t. $\forall i R_i \subseteq R$ (no new attributes)
  – Split F into $F_i$ s.t. $\forall i F$ entails $F_i$ (no new FDs)
  – $F_i$ involves only attributes in $R_i$

• Caveat: entirely possible to lose information
  – $F^+$ may entail FD X which is not in $(\bigcup_i F_i)^+$
    $\Rightarrow$ Decomposition lost some FDs
  – Possible to have $R \subseteq \bigotimes_i R_i$
    $\Rightarrow$ Decomposition lost some relationships

• Goal: minimize anomalies without losing info

We’ll revisit information loss later
### Splitting relations – example

**Consider the following relation:**

<table>
<thead>
<tr>
<th>Student Name</th>
<th>Student Email</th>
<th>Course</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSC333</td>
<td>Smith</td>
</tr>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSC444</td>
<td>Brown</td>
</tr>
<tr>
<td>Jaspreet</td>
<td>jaspreet@gmail</td>
<td>CSC333</td>
<td>Smith</td>
</tr>
</tbody>
</table>

**One possible decomposition**

- **Students(email, name)**
- **Taking(studentEmail, courseName)**
- **Courses(name, instructor)**
Gotcha: lossy join decomposition

- Consider a relation with one more tuple

<table>
<thead>
<tr>
<th>Student Name</th>
<th>Student Email</th>
<th>Course</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSC333</td>
<td>Smith</td>
</tr>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSC444</td>
<td>Brown</td>
</tr>
<tr>
<td>Jaspreet</td>
<td>jaspreet@gmail</td>
<td>CSC333</td>
<td>Smith</td>
</tr>
<tr>
<td>Mary</td>
<td>mary@gmail</td>
<td>CSC444</td>
<td>Rosenberg</td>
</tr>
</tbody>
</table>

- **Students Taking Courses** has bogus tuples!
  - Mary is not taking Brown’s section of CSC444
  - Xiao is not in Rosenberg’s section of CSC444

*Why did this happen? How to prevent it?*
Information loss with decomposition

• Decompose R into S and T
  – Consider FD a->b, with a only in S and b only in T

• FD loss
  – Attributes a and b no longer in same relation
    => Must join T and S to enforce a->b (expensive)

• Join loss
  – LHS and RHS no longer in same relation, no other connection
  – Neither (S ∩ T) -> S nor (S ∩ T) -> T in F⁺
    => Joining T and S produces bogus tuples (irreparable)

• In our example:
  – (email,course) ∩ {course,instructor} = {course}
  – course -/-> instructor and course -/-> email
Projecting FDs

• Once we’ve split a relation we have to refactor our FDs to match
  – Each FDs must only mention attributes from one relation

• Similar to geometric projection
  – Many possible projections (depends on how we slice it)
  – Keep only the ones we need (minimal basis)
FD projection algorithm

• Start with $F_i = \emptyset$

• For each subset $X$ of $R_i$
  - Compute $X^+$
  - For each attribute $a$ in $X^+$
    • If $a$ is in $R_i$
      - add $X \rightarrow a$ to $F_i$

• Compute the minimal basis of $F_i$

• Projection is expensive
  - Suppose $R_1$ has $n$ attributes
  - How many subsets of $R_1$ are there?
Making projection more efficient

• Ignore trivial dependencies
  – No need to add $X \rightarrow A$ if $A$ is in $X$ itself

• Ignore trivial subsets
  – The empty set or the set of all attributes (both are subsets of $X$)

• Ignore supersets of $X$ if $X^+ = R$
  – They can only give us “weaker” FDs (with more on the LHS)
Example: Projecting FD’s

• ABC with FD’s A->B and B->C
  – A⁺=ABC ; yields A->B, A->C
    • We ignore A->A as trivial
    • We ignore the supersets of A, AB⁺ and AC⁺, because they can only give us “weaker” FDs (with more on the LHS)
  – B⁺=BC ; yields B->C
  – C⁺=C ; yields nothing.
  – BC⁺=BC ; yields nothing.
Example -- Continued

• Resulting FD’s: $A \rightarrow B$, $A \rightarrow C$, and $B \rightarrow C$

• Projection onto $AC$ : $A \rightarrow C$
  – Only FD that involves a subset of $\{A, C\}$

• Projection on $BC$: $B \rightarrow C$
  – Only FD that involves subset of $\{B, C\}$
PART III: NORMAL FORMS
Motivation for normal forms

• Identify a “good” schema
  – For some definition of “good”
  – Avoid anomalies, redundancy, etc.

• Many normal forms
  – 1\textsuperscript{st}
  – 2\textsuperscript{nd}
  – 3\textsuperscript{rd}
  – Boyce-Codd
  – ... and several more we won’t discuss...

\textit{BCNF} \subseteq 3\textit{NF} \subseteq 2\textit{NF} \subseteq 1\textit{NF} (focus on 3\textit{NF}/BCNF)
1st normal form (1NF)

• No multi-valued attributes allowed
  – Imagine storing a list/set of things in an attribute
  => Not really even expressible in RA

• Counterexample
  – Course(name, instructor, [student,email]*)
  – Redundancy in non-list attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Instructor</th>
<th>Student Name</th>
<th>Student Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSCC43</td>
<td>Johnson</td>
<td>Xiao</td>
<td>xiao@gmail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jaspreet</td>
<td>jaspreet@utsc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mary</td>
<td>mary@utsc</td>
</tr>
<tr>
<td>CSCD08</td>
<td>Rosenberg</td>
<td>Jaspreet</td>
<td>jaspreet@utsc</td>
</tr>
</tbody>
</table>
2nd normal form (2NF)

- Non-prime attributes depend on candidate keys
  - Consider non-prime (ie. not part of a key) attribute ‘a’
  - Then $\exists$FD $X$ s.t. $X \rightarrow a$ and $X$ is a candidate key

- Counterexample
  - Movies($title, year, star, studio, studioAddress, salary$)
  - FD: $title, year \rightarrow studio; studio \rightarrow studioAddress; star \rightarrow salary$

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Star</th>
<th>Studio</th>
<th>StudioAddr</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>Hamill</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
<td>$100,000</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>Ford</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
<td>$100,000</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>Fisher</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
<td>$100,000</td>
</tr>
<tr>
<td>Patriot Games</td>
<td>1992</td>
<td>Ford</td>
<td>Paramount</td>
<td>Cloud 9</td>
<td>$2,000,000</td>
</tr>
<tr>
<td>Last Crusade</td>
<td>1989</td>
<td>Ford</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
<td>$1,000,000</td>
</tr>
</tbody>
</table>
3rd normal form (3NF)

- Non-prime attr. depend *only* on candidate keys
  - Consider FD X -> a
  - Either a \( \in X \) OR X is a superkey OR a is prime (part of a key)
  => No transitive dependencies allowed

- Counterexample:
  - studio -> studioAddr

  *(studioAddr depends on *studio* which is not a candidate key)*

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Studio</th>
<th>StudioAddr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
</tr>
<tr>
<td>Patriot Games</td>
<td>1992</td>
<td>Paramount</td>
<td>Cloud 9</td>
</tr>
<tr>
<td>Last Crusade</td>
<td>1989</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
</tr>
</tbody>
</table>
3NF, dependencies, and join loss

• Theorem: always possible to convert a schema to join-lossless, dependency-preserving 3NF

• Caveat: always possible to create schemas in 3NF for which these properties do not hold

• Join loss example 1:
  – MovieInfo(title, year, studioName)
  – StudioAddress(title, year, studioAddress)
  => Cannot enforce studioName \rightarrow studioAddress

• Join loss example 2:
  – Movies(title, year, star)
  – StarSalary(star, salary)
  => Cannot enforce Movies\bowtie StarSalary yields bogus tuples (irreparable)
Boyce-Codd normal form (BCNF)

• One additional restriction over 3NF
  – All non-trivial FD have superkey LHS

• Counterexample
  – CanadianAddress(street, city, province, postalCode)
  – Candidate keys: {street, postalCode}, {street, city, province}
  – FD: postalCode -> city, province

  – Satisfies 3NF: city, province both non-prime
  – Violates BCNF: postalCode is not a superkey
  => Possible anomalies involving postalCode

Do we care? How often do postal codes change?
Limits of decomposition

• Pick two...
  – Lossless join
  – Dependency preservation
  – Anomaly-free

• 3NF
  – Always allows join lossless and dependency preserving
  – May allow some anomalies

• BCNF
  – Always excludes anomalies
  – May give up one of join lossless or dependency preserving

*Use domain knowledge to choose 3NF vs. BCNF*